How Effective are OS-level Virtualization Tools for Managing Containers?

David Beserra, Robert Nantchouang, Mickael Chau, Marc Espie, Patricia Endo and Jean Araujo

Abstract As reliance on OS-level virtualization tools grows, understanding their efficiency in container management tasks is essential for optimizing performance. This study presents a comprehensive performance analysis of Docker, Podman, and LXD across key container management tasks: loading, starting, stopping, and removing containers and images. Our results indicate Docker's consistent superiority in speed, achieving the fastest execution times across tasks but at the cost of higher CPU usage. Podman demonstrates balanced resource efficiency, though generally slower than Docker in image loading. LXD, while slower in starting containers, exhibits lower CPU usage in parallel operations, making it suitable for scenarios where resource efficiency is prioritized over speed. These findings underscore the impact of tool choice on containerized environment performance, highlighting the importance of selecting a tool based on specific deployment requirements.

1 Introduction

OS-level virtualization technologies like Docker, Podman, and LXC have emerged as alternatives in response to the challenges from traditional virtualization techniques used in cloud computing environments, such as the inherent overheads associated with hypervisors responsible for managing VM activities and CPU resource

David Beserra, Robert Nantchouang, Mickael Chau, Marc Espie
École Pour l'Informatique et les Techniques Avancées (EPITA), Paris, France
e-mail: david.beserra@epita.fr

Patricia Takako Endo
Universidade de Pernambuco, Pernambuco (UPE), Brasil
e-mail: patricia.endo@upe.br

Jean Araujo
Universidade de Aveiro, Aveiro (UA), Portugal
e-mail: jean.araujo@ua.pt

allocation [4]. OS-level virtualization refers to a lightweight approach that allows multiple isolated user-space instances (containers) to run on a single host system, sharing the operating system kernel while maintaining isolation. Unlike traditional hypervisor-based virtualization, which requires separate virtual machines with full OS instances, OS-level virtualization offers efficiency by reducing overhead and enhancing resource utilization.

These tools aim to preserve the benefits of virtualization while addressing its performance issues [3] [2]. However, despite their recognized advantages, there is a pressing need to comprehensively evaluate the limitations and performance bottlenecks associated with these tools. Of particular concern are the performance implications of container management tasks, such as container starting, stopping, and removal. Different tools may exhibit varying performance in these activities due to distinct underlying mechanisms and architectures [11].

This paper evaluates the performance of Docker, Podman, and LXC/LXD OS-level virtualization tools within container management workflows. Our primary goal is to assess and compare the effectiveness of these tools across different container management activities. Specifically, we aim to determine the time required for container execution, stopping, and removal, quantify the resources necessary for these management tasks, and explore the impact of resource sharing on tool performance and host resource utilization. The insights from this analysis will support users and administrators in choosing the most efficient container management tools based on their performance and resource needs.

2 Related Works

The performance of OS-level virtualization tools has been extensively studied across a range of applications [9], however few studies have focused specifically on evaluating how these tools perform in their primary role: managing containers.

Inagaki et al. [8] analyzed Docker's performance in container management tasks within a microservice-based production environment. They explored the scalability of building and running containers, identifying bottlenecks in storage and network virtualization as well as issues related to kernel interfaces. However, their analysis was limited to Docker, reducing the applicability of their findings to other tools like Podman and LXD [3][7]. Additionally, the study did not quantify the resource requirements as the number of containers increased, leaving a gap in comprehensive performance analysis.

Straesser et al. [16] conducted an empirical evaluation involving 200,000 Docker Hub images to understand how different image configurations impacted container startup times. They found that no single configuration aspect solely dictated startup performance. However, the study focused exclusively on Docker and one specific management task—container starting—limiting its general relevance to broader container management operations or other OS-level tools.

Chhikara et al. [5] proposed an energy-efficient container migration scheme for IoT environments, addressing the computational overhead issues of traditional virtualization. While highlighting the advantages of containerization over virtual machines, the study did not compare the performance of multiple container management tools. This left an unaddressed question of how different OS-level tools perform under similar resource-constrained scenarios.

Pan et al. [14] explored the performance of orchestration engines, such as Kubernetes, Docker Swarm, and Apache Mesos, in microservice deployments. This study demonstrated Kubernetes' efficiency in complex deployments, but it did not extend to the direct performance analysis of OS-level tools like Docker, Podman, or LXD for basic container management tasks.

Vcilic et al. [6] focused on edge computing and the adaptability of orchestration platforms, assessing their ability to manage QoS under distributed resource conditions. While it provided valuable insights into orchestration at the edge, the study did not address the direct comparison of container management tools or basic container operations.

In a recent work, Melo et at. [11] compared Docker and Podman across various container management scenarios with different image sizes. The results indicated that Podman performed better with smaller containers, while Docker excelled with larger ones. However, this study did not evaluate the performance of these tools when a running service was present in the containers, nor did it assess the impact of concurrent resource sharing.

It is also important to distinguish that some studies equate container management with orchestration, which differs in focus. Orchestration involves decision-making processes for container allocation rather than the time taken for operations like starting or stopping containers [17] [15].

To the best of our knowledge, no current work has directly compared the performance of multiple OS-level tools in their core container management tasks or examined the impact of increased container density on resource utilization and performance. This study aims to address these gaps, using the methodology outlined in the following section.

3 Materials and methods

The main goal of this work is to evaluate and quantitatively compare the performance of OS-level virtualization tools in container management tasks. This goal is divided into specific objectives:

- Evaluate the time required by OS-level virtualization tools (Docker, Podman, LXC/LXD) to perform container image loading, container starting, container stopping, container removing, and container image removing.
- Measure the resources (specifically CPU, memory, and disk utilization) required by each tool for these container management tasks.

- Analyze the effect of resource sharing on both the tools' performance and the host system's resource utilization.

The first objective assesses how quickly each tool can complete key container management actions, from image loading to image removal. This analysis is crucial in environments where rapid provisioning or cleanup is required, as delays in these processes can impact overall system efficiency and scalability. The second objective, focused on resource requirements, is particularly relevant in high-scale environments where even minor increases in CPU, memory, or disk usage can lead to substantial cumulative costs. For example, a mere 5% increase in memory usage could necessitate additional gigabytes of RAM across multiple nodes. Lastly, the third objective examines how simultaneous container operations impact both tool performance and host stability. Understanding this effect is essential for efficient resource allocation in multi-container environments, as increased container density often intensifies resource competition.

3.1 Experimental Design

To meet these objectives, we designed a standardized container lifecycle management protocol (Figure 1) inspired by [11]. This protocol measures the performance of key stages in the container workflow, including: **Container Image Loading**, which involves loading the image required to initiate a container instance; **Container Starting**, which initializes the container to begin its designated tasks; **Container Stopping**, which halts the container process; **Container Removing**, which deletes the container instance from the system; and **Container Image Removing**, which frees up storage by removing the container image.

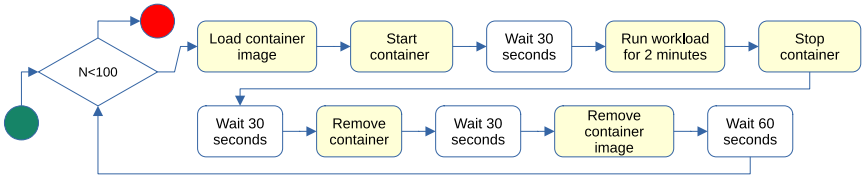


Fig. 1 Experimental Design Flow [11]

Each stage is followed by a cooldown period (30 seconds after starting, 60 seconds after image removal) to prevent interference between runs. This protocol repeats for up to 100 iterations to ensure consistent results. To simulate a real-world workload, a standardized container image running the High-Performance Linpack (HPL) benchmark [10] was employed. This benchmark solves a dense system of linear equations, producing a CPU- and memory-intensive workload. Configured to

run for at least two minutes, this workload provides a sustained stress test for container management. We empirically set the matrix size parameter N=8832 to meet the time requirement, using a single process to minimize resource interference.

To address the second and third objectives, we repeated the protocol with multiple containers running concurrently (2, 4, and 8 instances) to measure the impact of resource sharing on each tool's performance. Resource usage (CPU, memory, disk) and execution times were monitored to assess the effect of concurrent operations on each tool's efficiency.

3.2 Software Infrastructure: OS-level Virtualization Tools

The OS-level virtualization tools evaluated in this work were Docker, Podman, and LXC/LXD. Linux Containers (LXC) provide lightweight virtualization by sharing the host kernel, which minimizes overhead and allows for efficient resource use. LXC manages resource allocation through cgroups (Control Groups) and isolates processes using namespaces. However, to ensure comparability with Docker and Podman, both of which offer user-friendly interfaces and management capabilities, we use LXD, an extension of LXC. LXD enhances LXC with a more accessible interface and additional features, such as clustering and live migration, which make it suitable for complex deployment scenarios. This added functionality aligns LXD with Docker and Podman in terms of usability and management, providing a comparable experience across the tools while preserving LXC's lightweight nature [12].

Docker enables applications and dependencies to be packaged into container images, which can be quickly instantiated. It uses an overlay filesystem to enhance storage efficiency and supports image layering, facilitating the deployment of containers across cloud environments [13]. In turn, Podman is a daemonless container management tool, making it well-suited for environments prioritizing container independence and security, while maintains Docker compatibility[1]. The software versions tested were Docker 23.0.1, Podman 5.0.1, and LXD 5.21.

3.3 Resource Monitoring with Collectl

Throughout the experiments, Collectl was used to monitor system resources, capturing CPU, memory, and disk usage metrics in real time. Collectl's high sampling rate of 1 second allowed for fine-grained monitoring, ensuring that resource fluctuations, particularly under high-load conditions, were accurately recorded. This approach was essential for identifying transient peaks and dips in resource usage, providing a precise understanding of each tool's behavior and resource requirements in different container density scenarios. This data informs our analysis of each tool's efficiency and suitability for high-density, multi-container environments, where resource management is paramount.

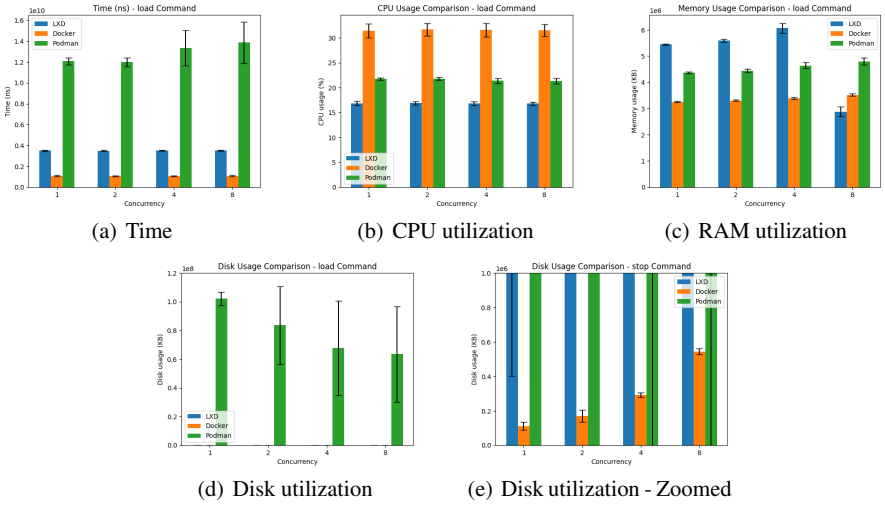
3.4 Physical Infrastructure

All experiments were conducted on an HP Compaq Elite server equipped with an Intel Core i7 3770 CPU at 3.4 GHz, 16 GB DDR3 RAM, and a 128 GB SSD, running Debian 12 Bookworm (kernel version 6.1-amd64).

4 Results

4.1 Container Image Loading

In the Container Image Loading operation, Docker consistently achieves the shortest loading times across all concurrency levels, indicating an efficient design optimized for rapid image handling and initialization, as shown in Figure 2. This high performance is facilitated by Docker's centralized daemon and overlay filesystem, which reduce redundancy in image loading. However, Docker's low loading time comes at the cost of high CPU usage, with CPU consumption increasing as concurrency rises. This trade-off suggests Docker's approach of allocating more CPU resources to minimize load times, making it ideal for CPU-rich environments where performance is prioritized.



(a) Time (b) CPU utilization (c) RAM utilization

(d) Disk utilization (e) Disk utilization - Zoomed

Fig. 2 Performance of OS-level virtualization tools when performing the Container Image Loading operation

LXD shows moderate loading times and significantly lower CPU usage, reflecting a conservative resource allocation approach. LXD's focus on CPU efficiency

aligns with its lightweight container architecture, but its slower loading times relative to Docker highlight a trade-off in prioritizing efficiency over speed. Podman, with the highest loading times across all concurrency levels, balances moderate CPU and low memory usage due to its daemonless architecture, which avoids centralized management overhead but sacrifices some speed in image handling. Minimal disk usage across all tools indicates that disk I/O is not a bottleneck in the loading process, though Podman's slower performance suggests a trade-off favoring lightweight resource usage over speed. The standard deviation in disk utilization is higher than in the other metrics, however this is a normal result of a performance and disk utilization measurement, regardless of the workload applied to the disk [2].

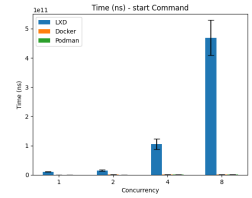
4.2 Container Starting

Figure 3 shows that the Container Starting operation further highlights Docker's performance-oriented design. Docker consistently maintains the shortest start times, with a notable increase in CPU usage at higher concurrency levels. This pattern underscores Docker's strategy of aggressively utilizing CPU resources to ensure rapid container initiation, making it suitable for environments requiring high performance and responsiveness. Podman achieves relatively low start times but shows a more balanced CPU usage profile, reflecting a moderate approach that optimizes resource usage while delivering reasonable performance.

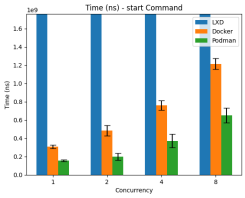
LXD, however, displays the longest start times, particularly at high concurrency, alongside the lowest CPU usage. This trade-off suggests that LXD's architecture prioritizes CPU efficiency over container start speed, potentially limiting its effectiveness in high-demand scenarios. Memory usage varies significantly, with LXD consuming the highest memory across all concurrency levels, likely due to its advanced management features like clustering and state management. Docker and Podman demonstrate efficient memory handling, aligning with their suitability for memory-constrained environments. These results show that while Docker and Podman achieve faster start times by allocating more CPU resources, LXD emphasizes resource efficiency, resulting in slower start times but lower CPU consumption.

4.3 Container Stopping

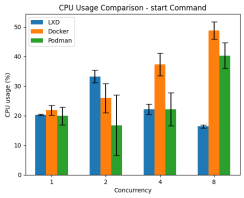
In the Container Stopping operation, Docker again outperforms in terms of execution time, consistently achieving the shortest stop times across concurrency levels with higher CPU usage, as shown in Figure 4. This finding aligns with Docker's focus on speed and makes it ideal for applications requiring rapid container termination. Podman's stop times are slightly longer than Docker's but remain low, with moderate CPU usage that scales with concurrency. This balance reflects Podman's design for efficient container management without excessive resource allocation.



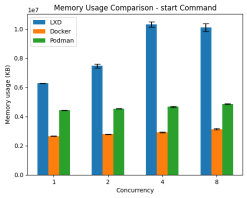
(a) Time



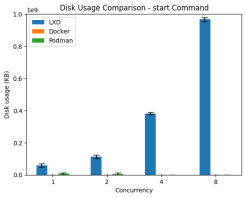
(b) Time - Zoom



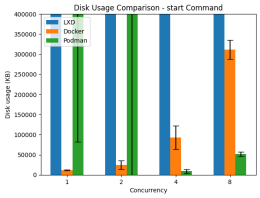
(c) CPU utilization



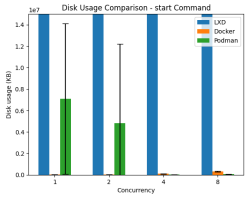
(d) RAM utilization



(e) Disk utilization



(f) Disk utilization - Zoom 1



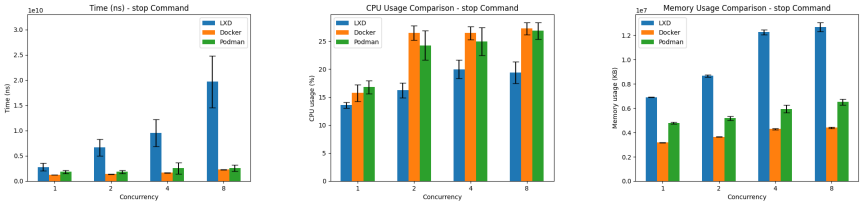
(g) Disk utilization - Zoom 2

Fig. 3 Performance of OS-level virtualization tools when performing the Start Container operation

LXD, by contrast, demonstrates the longest stop times and the lowest CPU usage, especially as concurrency increases, reflecting a conservative approach to resource management. LXD's high memory usage during stopping operations suggests that its extensive management capabilities, including handling metadata and preserving state, add overhead. This pattern suggests that LXD may be more appropriate in environments where CPU efficiency and container lifecycle management are more critical than rapid termination. Disk usage remains low across all tools for this operation, indicating that disk I/O does not significantly impact the container stopping process.

4.4 Container Removal

For the Container Removal operation, Docker again achieves the shortest removal times, underscoring its efficiency in managing container storage and state. Podman exhibits moderately low removal times, which slightly increase with higher concurrency, indicating efficient container handling even without a centralized daemon.

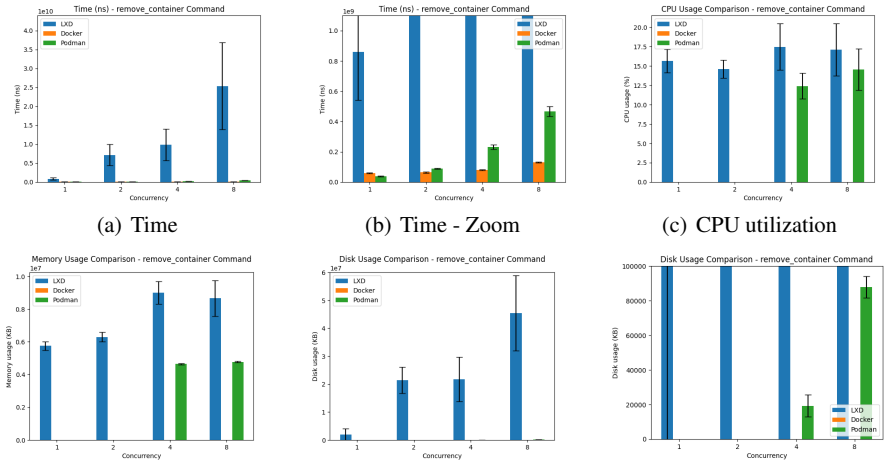


(a) Time - Full (b) CPU utilization (c) RAM utilization

(d) Disk utilization (e) Disk utilization - Zoomed

Fig. 4 Performance of OS-level virtualization tools when performing the Container Stopping operation

Podman's minimal CPU and disk usage for this operation suggest a balance between performance and resource conservation, making it suitable for environments with limited storage or compute resources.



(a) Time (b) Time - Zoom (c) CPU utilization

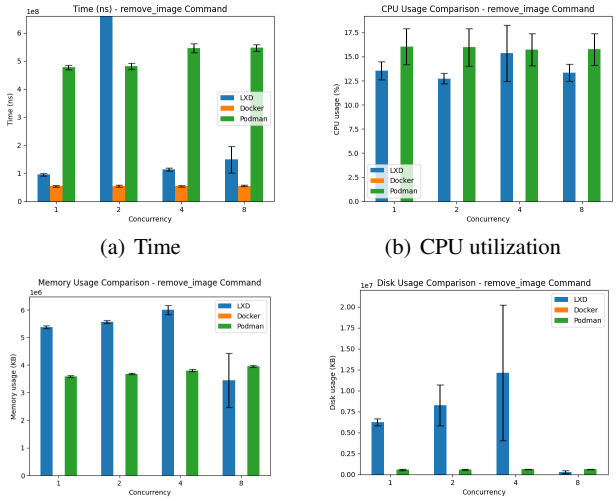
(d) RAM utilization (e) Disk utilization (f) Disk utilization - Zoomed

Fig. 5 Performance of OS-level virtualization tools when performing the Container Removal operation

LXD, however, shows consistently high removal times, especially at higher concurrency levels, along with elevated memory and disk usage. This performance profile suggests that LXD's comprehensive management of container metadata and persistent storage cleanup introduces additional overhead, which may not scale well in high-concurrency scenarios. The high disk usage in LXD suggests intensive interaction with container metadata and storage, potentially limiting its effectiveness in storage-constrained environments. These results suggest that Docker's rapid container removal capability is beneficial for environments requiring frequent container turnover, while LXD's resource-intensive approach may suit environments prioritizing complex container state management.

4.5 Container Image Removal

Figure 6 shows that the Container Image Removal operation further differentiates the tools' performance characteristics. Docker maintains the fastest image removal times, suggesting highly optimized image deletion protocols facilitated by its overlay filesystem and centralized resource management.



(a) Time (b) CPU utilization

(c) RAM utilization (d) Disk utilization

Fig. 6 Performance of OS-level virtualization tools when performing the Container Image Removal operation

LXD, however, displays substantial variation in image removal times, with an anomalously high execution time at concurrency level 2. This inconsistency, combined with high memory and disk usage, suggests that LXD's image management processes are resource-intensive, likely due to its advanced state management. Pod-

man, with moderate image removal times and balanced resource usage, indicates efficient but conservative resource handling.

5 Conclusions

This study evaluated the performance of three OS-level virtualization tools—LXD, Docker, and Podman—across key container management tasks, analyzing their execution times, CPU usage, memory usage, and disk usage at varying concurrency levels. Our results show that Docker consistently achieves the fastest execution times by leveraging high CPU usage, making it ideal for performance-critical environments with sufficient CPU resources. Podman balances moderate execution times with lower CPU and disk usage, aligning it with resource-constrained environments. LXD, while the slowest, prioritizes CPU efficiency and comprehensive management capabilities, making it suitable for scenarios where advanced container lifecycle management is essential.

The results highlight the trade-offs between speed, resource efficiency, and management capabilities in each tool's design. Docker's high CPU allocation ensures speed but may be costly in resource-limited setups, whereas Podman's balanced approach offers versatility. LXD's extensive management features come with higher memory and disk usage, impacting performance under high concurrency.

This work has limitations, including the lack of investigation into container orchestration tools like Kubernetes, which may impact each tool's efficiency in large-scale environments. Future research should explore the effects of orchestration, different hardware configurations, and network performance to broaden the understanding of these tools' applicability across diverse scenarios.

Based on the findings of this study, several future research directions can be explored to expand the understanding of OS-level virtualization tools for container management. One potential avenue is to investigate the integration of these tools within container orchestration frameworks, such as Kubernetes or Docker Swarm, to evaluate how orchestration impacts performance and resource utilization in complex, large-scale environments. Additionally, examining the effects of various hardware configurations, including multi-node clusters or specialized hardware such as GPUs, could provide insights into optimizing container management for specific workload types. Finally, exploring the impact of security features on the performance of these tools, particularly in environments with stringent compliance requirements, could further inform the selection of OS-level virtualization tools in security-sensitive scenarios.

References

1. Alqaisi, O.I., Tosun, A.Ş., Korkmaz, T.: Performance analysis of container technologies for computer vision applications on edge devices. IEEE Access (2024)
2. Beserra, D., Moreno, E.D., Endo, P.T., Barreto, J.: Performance evaluation of a lightweight virtualization solution for hpc i/o scenarios. In: 2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC). pp. 004681–004686 (Oct 2016). https://doi.org/10.1109/SMC.2016.7844970
3. Beserra, D., Pinheiro, M.K., Souveyet, C., Steffenel, L.A., Moreno, E.D.: Performance evaluation of os-level virtualization solutions for hpc purposes on soc-based systems. In: IEEE Int. Conf. on Advanced Information Networking and Applications (AINA 2017) (2017)
4. Beserra, D., Espie, M., Tomasimo, L., de Poncins, H., Lacombe, H., Vondracek, T., Araujo, J.: Could the topology of virtual processors affect the performance of a bsd-family os running in a vm? In: 2023 18th Iberian Conference on Information Systems and Technologies (CISTI). pp. 1–6. IEEE (2023)
5. Chhikara, P., Tekchandani, R., Kumar, N., Obaidat, M.S.: An efficient container management scheme for resource-constrained intelligent iot devices. IEEE Internet of Things Journal **8**(16), 12597–12609 (2020)
6. Čilić, I., Krivić, P., Podnar Žarko, I., Kušek, M.: Performance evaluation of container orchestration tools in edge computing environments. Sensors **23**(8), 4008 (2023)
7. Gomes Xavier, M., Veiga Neves, M., de Rose, F., Augusto, C.: A performance comparison of container-based virtualization systems for mapreduce clusters. In: Parallel, Distributed and Network-Based Processing (PDP), 22nd Euromicro Int. Conf. on. pp. 299–306 (2014)
8. Inagaki, T., Ueda, Y., Ohara, M.: Container management as emerging workload for operating systems. In: 2016 IEEE International Symposium on Workload Characterization (IISWC). pp. 1–10. IEEE (2016)
9. Li, Z., Kihl, M., Lu, Q., Andersson, J.A.: Performance overhead comparison between hypervisor and container based virtualization. In: 2017 IEEE 31st International Conference on advanced information networking and applications (AINA). pp. 955–962. IEEE (2017)
10. Luszczek, P., Meek, E., Moore, S., Terpstra, D., Weaver, V.M., Dongarra, J.: Evaluation of the hpc challenge benchmarks in virtualized environments. In: Euro-Par 2011: Parallel Processing Workshops. pp. 436–445. Springer (2012)
11. Melo, P., Gama, L., Dantas, J., Beserra, D., Araujo, J.: Performance evaluation of container management tasks in os-level virtualization platforms. In: 2023 IEEE International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE). pp. 1–6. IEEE (2023)
12. Moravcik, M., Segec, P., Kontsek, M., Uramova, J., Papan, J.: Comparison of lxc and docker technologies. In: 2020 18th International Conference on Emerging eLearning Technologies and Applications (ICETA). pp. 481–486. IEEE (2020)
13. Muzumdar, P., Bhosale, A., Basyal, G.P., Kurian, G.: Navigating the docker ecosystem: A comprehensive taxonomy and survey. Muzumdar, P., Bhosale, A., Basyal, GP, & Kurian, G.(2024). Navigating the Docker Ecosystem: A Comprehensive Taxonomy and Survey. Asian Journal of Research in Computer Science **17**(1), 42–61 (2024)
14. Pan, Y., Chen, I., Brasileiro, F., Jayaputera, G., Sinnott, R.: A performance comparison of cloud-based container orchestration tools. In: 2019 IEEE International Conference on Big Knowledge (ICBK). pp. 191–198. IEEE (2019)
15. Shih, W.C., Yang, C.T., Ranjan, R., Chiang, C.I.: Implementation and evaluation of a container management platform on docker: Hadoop deployment as an example. Cluster Computing **24**(4), 3421–3430 (2021)
16. Straesser, M., Bauer, A., Leppich, R., Herbst, N., Chard, K., Foster, I., Kounev, S.: An empirical study of container image configurations and their impact on start times. In: 2023 23rd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid). io (2023)
17. Yang, S., Wang, X., An, L., Zhang, G.: Yun: a high-performance container management service based on openstack. In: 2019 IEEE Fourth International Conference on Data Science in Cyberspace (DSC). pp. 202–209. IEEE (2019)